UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA - CCN
DEPARTAMENTO DE COMPUTAÇÃO – DC
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

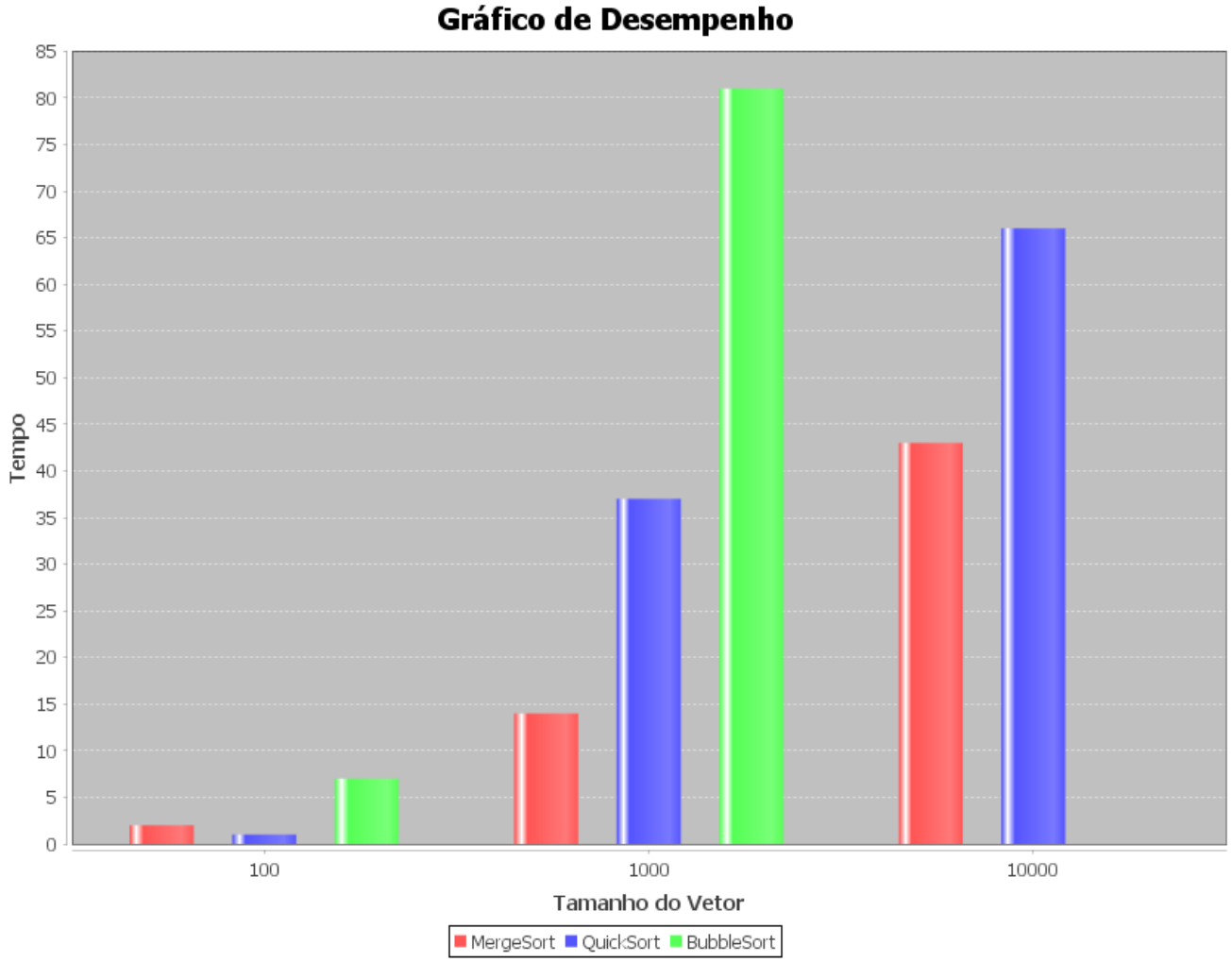
RELATÓRIO

Algorítmos de ordenação

- Análise sobre a eficiência dos algorítimos : MergeSort, QuickSort e BubbleSort. Verificando e analisando seu tempo de execução diante arrays de diferentes tamanhos.

PAULO EDUARDO RAMOS DE ARAUJO
Teresina – PI
2019

Os algorítimos estudados são : MergeSort, QuickSort e BubbleSort. A implementação dos algorítimos de ordenação foi feita em JAVA, utilizando suas bibliotecas e bibliotecas públicas.



✗ O gráfico não considera o tempo de execução do bubble sort para um array de 10000 nós pois seu tempo de execução estoura o gráfico.

O Quicksort se mostrou bastante efetivo quando executado em um array pequeno, porém o Merge tem maior desempenho para arrays maiores. O bubble sort, por ser um algorítimo mais simples e de fácil implementação, se mostrou tendo um tempo de execuçao que cresce rapidamente de acordo com o tamanho do array, diferente aos outros algorítimos.

O código baseaia-se nos algorítimos de ordenação encontrados no site da DevMedia, porém, os algorítimos do site funcionam para array de inteiros, por isso, foi desenvolvido uma classe com o método booleano (`CompOrderString(String a, String b)`) de comparação de duas strings que retorna *true* se a string A for primeiro, alfabeticamente, que a string B.

Ex. : `CompOrderString("Alicia", "Marcos");` <=> retorna *true*

Código:

```
package sortAlgorithm;

public class CompString {

	public static boolean CompOrderString(String a, String b) { // retorna true se a primeira String for primeiro

			// albafeticamente que a segunda string. TRUE = A <= B

		if (a == b)
			return false;
		if (a == null)
			return false;
		else if (b == null)
			return true;

		int i = 0; // index de percorrer o vetor
		String tam = a + b; // String auxiliar para criação do loop com a soma do tamanho das duas strings.

		a = a.toLowerCase();
		b = b.toLowerCase();

		for (int k = 0; k <= tam.length(); k++) {
			char aux_a;
			char aux_b;
			int ascii_a;
			int ascii_b;

			aux_a = a.charAt(i);
			aux_b = b.charAt(i);

			ascii_a = (int) aux_a;
			ascii_b = (int) aux_b;

			if (ascii_a < ascii_b) {
				return true;
			} else if (ascii_b < ascii_a) {
				return false;
			} else if (ascii_a == ascii_b) {
				i++;
				if (i >= a.length())
					return true;
				else if (i >= b.length())
					return false;
			}
		}
		return false;
	}
}
```